perplexity

make of pdf of all this qusetion with its answer in java language

I understand that you'd like a PDF containing all the common interview questions listed above, with their corresponding Java solutions (code and explanations).

However, as an AI text assistant, I cannot directly create or provide downloadable PDFs. But I can format all the questions and their Java solutions here, so you can easily copy and save them in your own document or PDF.

1. Two Sum

Problem:
Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to the target.

Solution (Java):

```
import java.util.*;

public class TwoSum {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[] { map.get(complement), i };
            }
            map.put(nums[i], i);
        }
        return new int[] {-1,-1}; // not found
    }
}
```

2. Reverse a Linked List

Problem:
Reverse a singly linked list.

Solution (Java):

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int val) { this.val = val; }
```

```
}

public class ReverseLinkedList {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        while (head != null) {
            ListNode next = head.next;
            head.next = prev;
            prev = head;
            head = next;
        }
        return prev;
    }
}
```

3. Detect Cycle in a Linked List

Problem:
Determine if a linked list has a cycle.

Solution (Java):

```
public class DetectCycle {
    public boolean hasCycle(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
            if (slow == fast) return true;
        }
        return false;
    }
}
```

4. Merge Intervals

Problem:
Merge all overlapping intervals.

Solution (Java):

```
import java.util.*;

public class MergeIntervals {
    public int[][] merge(int[][] intervals) {
        if (intervals.length <= 1) return intervals;
        Arrays.sort(intervals, Comparator.comparingInt(a -> a[0]));
        List<int[]> result = new ArrayList<>();
        int[] current = intervals[0];
        for(int i=1; i<intervals.length; i++){
            if(current[1] >= intervals[i][0]){
```

```
                current[1] = Math.max(current[1], intervals[i][1]);
            } else {
                result.add(current);
                current = intervals[i];
            }
        }
        result.add(current);
        return result.toArray(new int[result.size()][]);
    }
}
```

5. Valid Parentheses

Problem:
Determine if the input string is valid (matching brackets).

Solution (Java):

```
import java.util.*;

public class ValidParentheses {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
        for(char c : s.toCharArray()){
            if(c == '(' || c == '{' || c == '[')
                stack.push(c);
            else {
                if(stack.isEmpty()) return false;
                char open = stack.pop();
                if((c == ')' && open != '(') ||
                   (c == '}' && open != '{') ||
                   (c == ']' && open != '[')) return false;
            }
        }
        return stack.isEmpty();
    }
}
```

6. Minimum/Maximum in Rotated Sorted Array

Problem:
Find the minimum element in a rotated sorted array.

Solution (Java):

```
public class FindMinRotatedArray {
    public int findMin(int[] nums) {
        int left = 0, right = nums.length - 1;
        while(left < right){
            int mid = left + (right - left) / 2;
            if(nums[mid] > nums[right]){
                left = mid + 1;
```

```
            } else {
                right = mid;
            }
        }
        return nums[left];
    }
}
```

7. Find Duplicate Number

Problem:
Find the duplicate number in an array.

Solution (Java):

```
public class FindDuplicate {
    public int findDuplicate(int[] nums) {
        int slow = nums[0], fast = nums[0];
        do {
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while (slow != fast);
        slow = nums[0];
        while (slow != fast){
            slow = nums[slow];
            fast = nums[fast];
        }
        return slow;
    }
}
```

8. Longest Substring Without Repeating Characters

Problem:
Find the length of the longest substring without repeating characters.

Solution (Java):

```
import java.util.*;

public class LongestSubstring {
    public int lengthOfLongestSubstring(String s) {
        Set<Character> set = new HashSet<>();
        int left = 0, maxLen = 0;
        for(int right=0; right < s.length(); right++){
            while(set.contains(s.charAt(right))){
                set.remove(s.charAt(left++));
            }
            set.add(s.charAt(right));
            maxLen = Math.max(maxLen, right - left + 1);
        }
        return maxLen;
```

```
    }
}
```

9. Kth Largest Element in an Array

Problem:
Find the kth largest element in an array.

Solution (Java):

```
import java.util.*;

public class KthLargestElement {
    public int findKthLargest(int[] nums, int k) {
        PriorityQueue<Integer> minHeap = new PriorityQueue<>();
        for(int n : nums){
            minHeap.add(n);
            if(minHeap.size() > k){
                minHeap.poll();
            }
        }
        return minHeap.peek();
    }
}
```

10. Binary Tree Traversals (Inorder Example)

Problem:
Print inorder traversal of a binary tree.

Solution (Java):

```
import java.util.*;

class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int val) { this.val = val; }
}

public class BinaryTreeTraversal {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        Stack<TreeNode> stack = new Stack<>();
        TreeNode current = root;
        while (current != null || !stack.isEmpty()) {
            while (current != null) {
                stack.push(current);
                current = current.left;
            }
            current = stack.pop();
            result.add(current.val);
```

```
            current = current.right;
        }
        return result;
    }
}
```

If you want, you can copy this entire answer to a Word or Google Docs file, then export it as a PDF!

If you'd like an explanation for any of the problems or want more questions included, let me know!